|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 | 肖阳 | 学号 | 24300240055 | 第几次实验 | 14 |
| 实验代码（将代码粘贴在下面，如果有多题，将每一题的题目写在代码前面，加粗表示） | | | | | |
| #include <stdio.h>  #include <assert.h>  #include <math.h>  #define PI 3.14159265358979323846  // Represent a complex number x + yi.  struct complex\_number {    double x;    double y;  };  struct complex\_number zero={0.0,0.0};  // Returns a + b.  struct complex\_number add(struct complex\_number a, struct complex\_number b) {    struct complex\_number result;    result.x = a.x + b.x;    result.y = a.y + b.y;    return result;  }  // Returns a - b.  struct complex\_number minus(struct complex\_number a, struct complex\_number b) {    struct complex\_number result;    result.x = a.x - b.x;    result.y = a.y - b.y;    return result;  }  // Returns a \* b.  struct complex\_number mult(struct complex\_number a, struct complex\_number b) {    struct complex\_number result;    result.x = a.x \* b.x - a.y \* b.y;    result.y = a.x \* b.y + a.y \* b.x;    return result;  }  // Returns a / b, assuming b != 0.  // If b = 0, print an error message and return (0, 0).  struct complex\_number div(struct complex\_number a, struct complex\_number b) {    struct complex\_number result;    result.x = (a.x \* b.x + a.y \* b.y) / (b.x \* b.x + b.y \* b.y);    result.y = (a.y \* b.x - a.x \* b.y) / (b.x \* b.x + b.y \* b.y);    return result;  }  // Returns the absolute value (or modulus) of a = x + yi, which is  // sqrt(x^2 + y^2).  double modulus(struct complex\_number a) {    return sqrt(a.x \* a.x + a.y \* a.y);  }  // Returns the square root of a.  void square\_root(struct complex\_number a, struct complex\_number\* x\_1, struct complex\_number\* x\_2) {    double magnitude = sqrt(a.x \* a.x + a.y \* a.y);    double angle = atan2(a.y, a.x);      // First root    x\_1->x = sqrt(magnitude) \* cos(angle / 2);    x\_1->y = sqrt(magnitude) \* sin(angle / 2);      // Second root (with opposite sign)    x\_2->x = sqrt(magnitude) \* cos(angle / 2 + PI);    x\_2->y = sqrt(magnitude) \* sin(angle / 2 + PI);  }  // Returns a^n.  struct complex\_number power(struct complex\_number a, int n) {    double magnitude = modulus(a);    double angle = atan2(a.y, a.x);    double result\_angle = angle \* n;    double result\_magnitude = pow(magnitude, n);    struct complex\_number result;    result.x = result\_magnitude \* cos(result\_angle);    result.y = result\_magnitude \* sin(result\_angle);    return result;  }  // Compute the two roots of quadratic equation ax^2 + bx + c = 0.  // Let |x\_1| <= |x\_2|.  void solve\_quadratic(struct complex\_number a, struct complex\_number b, struct complex\_number c,    struct complex\_number\* x\_1, struct complex\_number\* x\_2) {    // Calculate discriminant: b² - 4ac    struct complex\_number four = {4.0, 0.0};    struct complex\_number two = {2.0, 0.0};    struct complex\_number discriminant = minus(mult(b, b), mult(four, mult(a, c)));      // Calculate square root of discriminant    struct complex\_number sqrt\_disc[2];    square\_root(discriminant, &sqrt\_disc[0], &sqrt\_disc[1]);      // Calculate roots: (-b ± √(discriminant)) / (2a)    struct complex\_number neg\_b = mult(b, (struct complex\_number){-1.0, 0.0});      // First root    \*x\_1 = div(add(neg\_b, sqrt\_disc[0]), mult(two, a));      // Second root    \*x\_2 = div(add(neg\_b, sqrt\_disc[1]), mult(two, a));  }  // Compare whether two complex numbers are equal (considering floating-point errors)  int complex\_equal(struct complex\_number a, struct complex\_number b, double epsilon) {    return (fabs(a.x - b.x) < epsilon) && (fabs(a.y - b.y) < epsilon);  }  // defined a small epsilon value for floating-point comparisons  const double EPSILON = 1e-6;  void TestAdd(){    // test 1：add    struct complex\_number a = {1.0, 2.0};    struct complex\_number b = {3.0, 4.0};    struct complex\_number expected = {4.0, 6.0};    struct complex\_number result = add(a, b);    assert(complex\_equal(result, expected, EPSILON));    printf("TestCase add passed!\n");  }  void TestMinus(){    // test 2：minus    struct complex\_number a = {5.0, 6.0};    struct complex\_number b = {3.0, 4.0};    struct complex\_number expected = {2.0, 2.0};    struct complex\_number result = minus(a, b);    assert(complex\_equal(result, expected, EPSILON));    printf("TestCase minus passed!\n");  }  void TestMult(){    // test 3：mult    struct complex\_number a = {1.0, 2.0};    struct complex\_number b = {3.0, 4.0};    struct complex\_number expected = {-5.0, 10.0};    struct complex\_number result = mult(a, b);    assert(complex\_equal(result, expected, EPSILON));    printf("TestCase mult passed!\n");  }  void TestDiv(){    // test 4：div    struct complex\_number a = {1.0, 2.0};    struct complex\_number b = {3.0, 4.0};    struct complex\_number expected = {0.44, 0.08}; // 近似值    struct complex\_number result = div(a, b);    assert(complex\_equal(result, expected, 0.1)); // 使用较大的epsilon    printf("TestCase div passed!\n");  }  void TestModulus(){    // test 5：modulus    struct complex\_number a = {3.0, 4.0};    double expected = 5.0;    double result = modulus(a);    assert(result == expected);    printf("TestCase modulus passed!\n");  }  void TestSquareRoot1(){    // test 6：square\_root    struct complex\_number a = {3.0, 4.0};    struct complex\_number x\_1 = {0.0, 0.0};    struct complex\_number x\_2 = {0.0, 0.0};    square\_root(a,&x\_1,&x\_2);    // (2 + 1i)^2 = 3 + 4i    struct complex\_number expectedx\_1 = {2.0, 1.0};    struct complex\_number expectedx\_2 = {-2.0, -1.0};    assert(complex\_equal(x\_1, expectedx\_1, EPSILON) && complex\_equal(x\_2, expectedx\_2, EPSILON) ||           complex\_equal(x\_1, expectedx\_2, EPSILON) && complex\_equal(x\_2, expectedx\_1, EPSILON));    printf("TestCase square\_root passed!\n");  }  void TestSquareRoot2(){    // test 7：square\_root    struct complex\_number a = {-4.0, 0.0};    struct complex\_number x\_1 = {0.0, 0.0};    struct complex\_number x\_2 = {0.0, 0.0};    square\_root(a,&x\_1,&x\_2);    // sqrt(-4) = 0 + 2i 或 0 - 2i    // 根据实现，应该返回x\_1 = 0 + 2i , x\_2 = 0 - 2i    struct complex\_number expectedx\_1 = {0.0, 2.0};    struct complex\_number expectedx\_2 = {0.0, -2.0};    assert(complex\_equal(x\_1, expectedx\_1, EPSILON) && complex\_equal(x\_2, expectedx\_2, EPSILON));    printf("TestCase square\_root passed!\n");  }  void TestPower1(){    // test 8：power    struct complex\_number a = {1.0, 1.0};    int n = 3;    struct complex\_number expected = {-2.0, 2.0}; // (1+i)^3 = -2 + 2i    struct complex\_number result = power(a, n);    assert(complex\_equal(result, expected, EPSILON));    printf("TestCase power passed!\n");  }  void TestPower2(){    // test 9：power    struct complex\_number a = {5.0, -3.0};    int n = 0;    struct complex\_number expected = {1.0, 0.0};    struct complex\_number result = power(a, n);    assert(complex\_equal(result, expected, EPSILON));    printf("TestCase power passed!\n");  }  void TestPower3() {    // test 10：power    struct complex\_number a = {1.0, 1.0};    int n = -1;    struct complex\_number expected = {0.5, -0.5}; // 1/(1+i) = (1 - i)/2    struct complex\_number result = power(a, n);    assert(complex\_equal(result, expected, EPSILON));    printf("TestCase power passed!\n");  }  void TestSolveQuadratic1() {    // test 11：solve\_quadratic    // 方程 x^2 - 3x + 2 = 0，根为 x=1 和 x=2    struct complex\_number coef\_a = {1.0, 0.0};    struct complex\_number coef\_b = {-3.0, 0.0};    struct complex\_number coef\_c = {2.0, 0.0};    struct complex\_number x1, x2;    solve\_quadratic(coef\_a, coef\_b, coef\_c, &x1, &x2);    struct complex\_number expected1 = {1.0, 0.0};    struct complex\_number expected2 = {2.0, 0.0};    assert(complex\_equal(x1, expected1, EPSILON) && complex\_equal(x2, expected2, EPSILON) ||           complex\_equal(x1, expected2, EPSILON) && complex\_equal(x2, expected1, EPSILON));    printf("TestCase solve\_quadratic passed!\n");  }  void TestSolveQuadratic2() {    // test 12：solve\_quadratic    // 方程 x^2 + (3+2i)x + (1-4i) = 0    struct complex\_number a = {1.0, 0.0};    struct complex\_number b = {3.0, 2.0};    struct complex\_number c = {1.0, -4.0};    struct complex\_number x1, x2;    solve\_quadratic(a, b, c, &x1, &x2);    // 手动计算根：    // 判别式 = (3+2i)^2 - 4\*(1)\*(1-4i) = 9 + 12i + 4i^2 - 4 + 16i = (9 - 4 - 4) + (12 + 16)i = 1 + 28i    // sqrt(1 + 28i) ≈ 5.2915 + 2.3670i （近似值）    // 根 = [-(3+2i) ± (5.2915 + 2.3670i)] / 2    // x1 ≈ ( -3 -2i +5.2915 +2.3670i ) / 2 ≈ (2.2915 +0.3670i) / 2 ≈ 1.1458 +0.1835i    // x2 ≈ ( -3 -2i -5.2915 -2.3670i ) / 2 ≈ (-8.2915 -4.3670i) / 2 ≈ -4.1458 -2.1835i    // 为了测试，假设 x1 = 1 + 0i 和 x2 = -4 + 0i 作为示例（实际应根据计算结果）    // 这里为了简单起见，检查乘积和和是否符合    struct complex\_number expected\_sum = b; // x1 + x2 = -b/a = -(3+2i)/1 = -3-2i    struct complex\_number expected\_product = c; // x1 \* x2 = c/a = 1-4i    struct complex\_number sum = add(x1, x2);    struct complex\_number product = mult(x1, x2);    assert(complex\_equal(sum, (struct complex\_number){-3.0, -2.0}, EPSILON));    assert(complex\_equal(product, (struct complex\_number){1.0, -4.0}, EPSILON));    printf("TestCase solve\_quadratic passed!\n");  }  void TestCase(){    TestAdd();    TestMinus();    TestMult();    TestDiv();    TestModulus();    TestSquareRoot1();    TestSquareRoot2();    TestPower1();    TestPower2();    TestPower3();    TestSolveQuadratic1();    TestSolveQuadratic2();    printf("All testcases passed!\n");  }  int main() {    TestCase();    return 0;  } | | | | | |
| 运行结果（截图运行结果，如果有多题，写清题目加粗表示，再粘贴截图） | | | | | |
|  | | | | | |